Just-in-time Staging of Large Input Data for Supercomputing Jobs

Henry M. Monti and Ali R. Butt
Virginia Tech
Blacksburg, VA 24061
Email: {hmonti, butta}@cs.vt.edu

Sudharshan S. Vazhkudai
Oak Ridge National Laboratory
Oak Ridge, TN 37831
Email: vazhkudaiss@ornl.gov

***Abstract*—High performance computing is facing a data deluge from state-of-the-art colliders and observatories. Large data-sets from these facilities, and other end-user sites, are often inputs to intensive analyses on modern supercomputers. Timely staging in of input data at the supercomputer's local storage can not only optimize space usage, but also protect against delays due to storage system failures. To this end, we propose a *just-in-time staging* framework that uses a combination of batch-queue predictions, user-specified intermediate nodes, and decentralized data delivery to coincide input data staging with job startup. Our preliminary prototype has been integrated with widely used tools such as the PBS job submission system, BitTorrent data delivery, and Network Weather Service network monitoring facility.**

I. Introduction

The advent of Petaflop computers, observatories and large-scale colliders are pushing the envelope on data-set sizes. For instance, the Large Hadron Collider (LHC) at CERN will generate petabytes of data, which scientists will analyze to glean insights into the origin of the universe. Similarly, the Spallation Neutron Source (SNS) at Oak Ridge National Laboratory (ORNL) will generate hundreds of terabytes of data that will be analyzed by users from a variety of domains ranging from medicine to engineering. These large input data-sets are processed by a geographically dispersed user base, often times, on large-scale supercomputers referred to as leadership class facilities. Therefore, result output data from supercomputer simulations are not the only source that is driving data-set sizes. Input data sizes are also growing many fold.

A typical job work-flow comprises of staging the input data from the end-user location onto the supercomputer's parallel file system called scratch space, and submitting the job to the scheduler at the center. From then on, the job waits in the queue for its turn, while the input data waits on the scratch space. HPC centers are heavily crowded and it is not uncommon for a job to spend hours—or even days on end—in the queue. Consequently, the turnaround time of jobs in popular HPC centers is much higher than their actual runtime. This problem is so acute in some centers that funding agencies (DOD, NSF, DOE) are already requesting performance specification metrics such as *expansion factor* in addition to the traditional job turnaround time metric. The expansion factor is defined as the ratio *(wall_time + wait_time)/wall_time* averaged over all jobs (the closer to 1, the better).

The time a job takes to complete, i.e., *(wall_time + wait_time)*, is also the time the input data spends in the scratch parallel file system. The scratch parallel file system is an expensive commodity, and provisioning and maintaining it is usually a notable fraction of the HPC center's operations budget. The scratch space is meant for storing data for currently running or soon to run jobs. From a center standpoint, sub-optimal use of the scratch resource could impact the center's serviceability, i.e., the ability to serve more incoming jobs. From a user standpoint, the input data is exposed to potential unavailability due to storage system failure while it is waiting for the job to be scheduled. Storage system failure has been slated as one of the primary reasons for system downtime in many leadership class machines [7], [13], [16]. Consequently, when the job is ready to run, crucial pieces of input data may be unavailable, requiring a rescheduling. This can result in undesirable delays.

To address these issues, we propose a framework for just-in-time staging of input data-sets associated with supercomputing jobs. Our framework attempts to have the data available at the scratch parallel file system, from the end-user location, just before the job is about to run, thereby mitigating many of the aforementioned issues. The basic idea behind just-in-time staging is as follows. Using our framework, a user first obtains an *advisory opinion* from a batch queue prediction service [1] at the center, which gives an estimate, with some confidence, of when the job will run. Based on this advice, the user then submits his job script to a *staging manager* that initiates the data staging in a globally (center-wide) optimal fashion. In the meantime, the compute job is submitted to the batch scheduler and is setup to commence only after the input data is staged. The staging manager uses a number of intermediate storage locations in the data path between the end-user and the HPC center to setup a peer-to-peer overlay. It uses a combination of high-efficiency data dissemination (using BitTorrent [3]) and network monitoring (using Network Weather Service (NWS) [15]) to exploit orthogonal, residual bandwidth and

This research is sponsored in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725, and by the U.S. National Science Foundation Faculty Early Career Development (CAREER) Program CCF-0746832.

to dynamically adapt to network volatility, respectively. Such dynamic adaptation allows the manager to perform just-in-time data staging to meet the job's commencement schedule.

Thus, our approach is able to use HPC center resources in a judicious fashion by not staging the data too early, and is also able to protect user data from potentially undesirable failure scenarios.

II. Problem Space

In this section, we discuss the issues involved in designing a just-in-time staging solution for an HPC center. In order to stage the data to be coincident with job startup, we need intelligent estimates of the following. First, we need to know when the user's job will commence. Predicting job start times is a well explored area of research. Several studies [12], [4] exist that use a combination of job requirements such as number of processors, run time, and previous and current behavior of the batch scheduler to predict, with reasonable accuracy, the time at which a job would be started. In our work, we will not delve into the intricacies of designing a prediction service and use an existing service, NWS [15], developed at the University of California at Santa Barbara. Instead, we focus our efforts on using such a service to design a data staging scheme. The batch queue prediction service from NWS provides two kinds of information: (i) given job characteristics, it predicts a statistical upper bound on how long the job will wait prior to execution; (ii) given job characteristics and a start deadline, it can give a probability of the job starting by the deadline [1]. Both of these predictions are relevant to our purposes.

Second, we need an estimate of how long the data staging would take from the end-user site to the HPC center. Much like estimating job start times, predicting wide-area data transfer is a whole body of research in itself. Our own previous work dealt with predicting large GridFTP transfer times in a data grid environment using regression techniques that combine previous history of transfers with current network bandwidth measurements [14]. In our target setting, however, we propose to use intermediate storage locations in the data path between the center and the user by arranging them in a tree structure and using BitTorrent for data dissemination. Such an approach entails continuous bandwidth measurements for the network links connecting the center, the intermediate nodes, and the end-user. Further, the changing network bandwidth values need to be factored in to revise the route dynamically in an attempt to stage the input data before the job startup deadline.

III. Design

The just-in-time staging of job input data ensures timely delivery of the data at the HPC center and is achieved using a combination of strategies both at the center as well as the end-user site. In our design, the HPC center provides a batch queue prediction service (e.g., NWS batch queue prediction [1]), which the users can query before submitting their jobs to get an estimate of queue wait times. This is a reasonable requirement, as such a service is common in modern HPC sites. For example, each of the nine TeraGrid [6] supercomputer centers run a service to furnish job wait time estimates, which the users can use to reduce turnaround times for their jobs.

As noted earlier, the prediction service can usually provide both wait time estimates as well as a probability for a job starting by a given user deadline. In cases where direct wait time predictions are unavailable, the user needs to pose a query with a deadline for when the job should start by. For example, the query for the NWS batch prediction takes the following values: cluster, queue, number of nodes, runtime and deadline. For our purposes, the percent probability result can then be used to determine the potential job wait time for that queue. A 90% or higher probability can be treated as an affirmation of the user's deadline and can be used as the time when the job will be started. The job can potentially start earlier than this predicted deadline due to inaccuracies in the prediction or due to failure of other running jobs. While a lower probability may mean that the job may not commence by the user-specified deadline, it is only an estimate. To accommodate this, we can let the user tweak the estimate by some factor, f. Here, the user adjusts the estimated deadline, obtained from the batch queue prediction service, by a percentage to denote a tighter job start deadline. A user might do this for one of two reasons. First, the user may wish to use the prediction with "guarded optimism" to account for earlier job starts. Second, he might wish to finish staging the data as early as possible by using an artificial tighter deadline, thereby shifting the burden of protecting the data during the prolonged wait time to the center. Therefore, allowing f to be large can unduly affect other jobs, which have genuine tight deadlines. Limiting the adjustment to only a factor is necessary to ensure that there is global fairness in staging towards all jobs. Consequently, the prediction service will also report its estimate to a *staging manager* at the center so that the manager can ensure that the staging deadline submitted by the user is within the factor.

The user then submits a job script to the staging manager at the center with a description of the job and other details necessary for just-in-time staging. These include attributes such as the job startup deadline, $T_{JobStartup}$ (mentioned above), a set of intermediate nodes, $< N_i, P_i >$, where P_i denotes the usage properties of the intermediate nodes N_i, for the decentralized staging process, and the size of the input data-set, S. The staging manager also takes as input the current snapshot, BW_i, of the observed NWS bandwidth between the HPC center and N_i as well as between the N_i's themselves. Based on these parameters, the manager decides upon either a direct or a decentralized transfer of the job's input data. The decision, which we call an input data staging schedule, I_s, delivers the data in time, T_{Stage}, which satisfies the property, $T_{Stage} \leq T_{JobStartup}$.

Even after a particular course of action, e.g., decentralized transfer, is chosen, a decision-making component constantly re-evaluates the data staging based on an updated $< N_i, P_i, BW_i' >$, where BW_i' is the latest snapshot of NWS bandwidth measurements. If the re-evaluated time to staging, T'_{Stage}, satisfies the property, $T'_{Stage} > T_{JobStartup}$,

then, alternate routes are taken to stage the data before job execution. However, this is only feasible if such routes are available at the time of re-evaluation. To accommodate this, we periodically re-evaluate the decentralized transfer during the entire course of the stage-in to see if newer optimal routes have become available. This would enable us to meet the stage-in deadline and to ensure that the job scheduler is not starved with no job to schedule due to an unfinished stage-in.

The user submits the job script simultaneously to the staging manager and the batch queue, so that, while data staging starts, the job gets in line in the queue and the compute part can start by the user deadline. The staging manager then submits the decentralized data staging job to a data job queue and sets up a dependency such that the compute job does not begin until the staging-in task has finished. To this end, we use and extend our earlier works [16], [9] on instrumenting the job submission system, the *stagesub* tool that is being used in the No.5 supercomputer in Top500, ORNL Jaguar. Having a center-wide staging manager has the advantage that the manager can perform global optimization (e.g., higher priority to a stage-in that is on a tight deadline).

A final piece in the just-in-time data staging architecture is the utilization of a number of user-specified intermediate nodes (discussed below), using which data can be staged into the center.

A. Intermediate Nodes

Our system uses a number of intermediate nodes (N_is) that can provide temporary storage for data on the path from the client site to the HPC center. While submitting a job the client also provides to the center a set of nodes that can be used as intermediate nodes. These nodes can be the client's own collaborating sites, from where other input data can also be staged. This has an added advantage of letting the HPC center asynchronously retrieve data from other sources, decoupled from the client site. The intermediate nodes provide multiple data flow paths from the submission site to the center, which lead to better bandwidth utilization, faster staging speeds, as well as fault-tolerance in the face of failures.

To account for a case where a client does not have enough intermediate nodes for efficient stage-in, we can rely on the idea of *Landmark nodes* [9], a number of geographically distributed nodes that are always available and can serve as intermediate nodes. The Landmark nodes can be other HPC centers, or nodes along national links such as, Internet2, Lambda Rail or the TeraGrid to which many end-users may be connected.

B. Supporting Just-in-Time Staging

Once the submission site receives an estimate from the batch queue prediction service, the data staging process is initiated. First, the center chooses a number of nodes from the set of N_i's ordered by available bandwidth. The exact number of nodes used for this purpose, i.e., the fan-out, is chosen to stage-in all the necessary data before the predicted job start time. These chosen N_i's serve as the Level-1 intermediate nodes. Note that the selected fan-out is not static, and can vary depending on the actual transfer speeds. Second, the input data is split into chunks and parallel transfer of the chunks to Level-1 nodes is initiated. The transfer may also involve further levels of intermediate nodes (up to Level-N). Alternatively, depending on the availability of intermediate nodes, the client can also stage the data to Level-N nodes much earlier than the deadline.

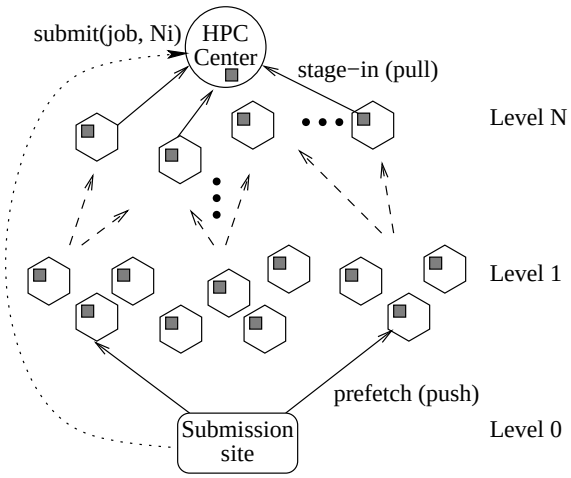

Fig. 1. The data flow path from the submission site to the HPC center. Each intermediate node (hexagon) run NWS (gray square) for bandwidth monitoring.

As the job startup deadline approaches, the close proximity of the Level-N nodes to the center allows them to quickly move the input data to the center's scratch space. Also, this design allows the Level-N nodes to stage-in the data at peak (pre-specified) bandwidth at the most appropriate time without worrying about the availability (and connection speed) of the submission site. This process is illustrated in Figure 1.

The use of intermediate nodes in our system provides multiple data-flow paths from the submission site to the center, leading to several alternative options for data delivery. For instance, data may be replicated across different N_i's during the transfer from one level to the other. This will allow the center to pull data from a number of locations, thus providing fault tolerance against node failure, as well as better utilization of the available in-bandwidth at the center. The schedule can also be used to simultaneously deliver data to multiple interested sites in the network.

Since, our design allows the submission site to push the data onto intermediate nodes, we employ implicit replication of the data by sending it to more than a single intermediate node. In addition to this, we can also apply erasure code [8], [11] to improve the reliability of the transfer, while minimizing the amount of transferred data.

Another aspect of the staging-in process is to ensure that the necessary data is copied to the center scratch space in time, and potential delays in the job startup are avoided. Given the dynamically changing bandwidths between participants, a fixed or statically chosen fan-out/data path is insufficient.

```
#PBS -N myjob
#PBS -l nodes=128, walltime=12:00
mpirun -np 128 ~/MyComputation
#Stagein file://SubmissionSite:/home/user/input1
file:///home/scratch/user/input1
#InterNode node1.Site1:49665:50GB
...
#InterNode nodeN.SiteN:49665:30GB
#Deadline 1/14/2007:12:00
```

Fig. 2. An example instrumented PBS script.

Therefore, we employ NWS [15] to monitor and estimate the available bandwidth between participating nodes. The center uses this information to decide whether a chosen fan-out is sufficient to meet a particular deadline, or needs to be increased. Finally, this decision is re-evaluated at each level to ensure proper staging.

IV. Implementation

We have implemented our just-in-time staging manager using about 2500 lines of C code, and used FreePastry [5] as the p2p overlay.

A. Integration with Job Submission

We instrumented the widely-used PBS [2] job submission system to let the users specify intermediate nodes and deadlines. An example instrumented PBS script is shown in Figure 2, where the user specifies intermediate nodes and deadlines as well as details such as available storage capacities.

The annotated script is submitted to our parser on the HPC center. The parser filters out the staging-specific directives and passes them to the staging manager. The remaining script is forwarded to the standard PBS queue.

B. Integration with BitTorrent and NWS

We use NWS [15] to track statistics, e.g., available bandwidth, for each intermediate node. These measurements are used for adjusting fan-out to enable staging of data in time.

We exploit BitTorrent's [3] scatter-gather protocol for transferring data by extending the protocol to use NWS bandwidth measurements. This allows efficient use of the orthogonal bandwidth, and provides opportunities to improve overall transfer times. The *Staging Manager* creates a "torrent" file for the subset of data to be transmitted to a set of chosen intermediate nodes. Upon receiving the torrent file, the nodes use the metadata information in the file along with a BitTorrent tracker to "download" the data subset to their local storage. The process is repeated at all the intermediate node levels. When the job is about to run at the center, the Manager can use appropriate torrent files to pull the input data from the intermediate nodes to the center, thus completing the stage-in process.

V. Evaluation

In this section, we present an evaluation of our just-in-time data staging approach using the implementation of Section IV.

TABLE I
Transfer times (in seconds) using a direct transfer (scp) and our decentralized staging.

File Size	100 MB	240 MB	500 MB	2.1 GB
Direct	172	351	794	3082
Client offbad	139	258	559	2164
Pull	43	106	193	822

TABLE II
The time to transfer a 2.1 GB file using standard BitTorrent. The equivalent phases for our scheme are shown in brackets.

Phase	Time(s)
Send to intermediate nodes (Client offbad)	2653
HPC Center download (Pull)	960

A. Experimental Setup

We use the PlanetLab [10] testbed for our experiments. To this end, we chose six geographically distributed sites, and arranged them in a tree structure with the submission site as the root, and with only one level of four intermediate nodes. In the following, the reported numbers represent averages over a set of three runs.

For this initial investigation, we focus on the ability of the presented design to achieve its goal of decentralized stage-in, and its ability to reduce data transfer times as a proof-of-concept. The use of PlanetLab limit our evaluation in terms of the size of the data and volatility of used resources. In true HPC systems, the data sizes will be much larger and resources more stable. Evaluating our just-in-time staging on such a scale remains the goal of our current and future research.

B. Decentralized Stage-in vs. Direct Transfer

In this experiment, we determine the feasibility of our approach compared to a simple point-to-point direct transfer using scp. For this purpose, we used a range of file sizes from 100 MB to 2.1 GB. We measured the time of a direct transfer between the submission site and the center, as well as the transfer times for the proposed just-in-time staging. Table I shows the results of the time it takes for the data to be transferred from the client to HPC center directly (Direct), from client to Level-1 nodes (Client offload), and from Level-1 to the Center (Pull). Compared to a direct transfer, the decentralized stage-in can reduce the last-hop transfer times by 69.8% to 75.7% for 240 MB and 500 MB data sizes, respectively. This can lead to improved HPC center serviceability by reducing the time the scratch space has to hold data for a job that will run in distant future.

C. Effect of Using NWS Measurements

Next, we compare our NWS-based transfer approach with a regular BitTorrent-based data transfer. In this case, we use NWS bandwidth measurements to greedily provision Level-1 nodes to increase the fan-out to utilize the maximum client outbound bandwidth. Table II shows the time taken to deliver a 2.1 GB data-set using the regular, unmodified BitTorrent protocol. Compare these to the transfer times using our just-in-time staging shown earlier in Table I. The results indicate that

both Client offload and Pull in our approach out-perform the corresponding steps in regular BitTorrent transfer. The Client offload to Level-1 nodes is 18.4% faster. The time to pull the file to the Center scratch space is improved significantly by 14.4%. These results show that bandwidth measurement provides a good tool for improving stage-in times.

D. When to Use Decentralized Stage-in?

In the above experiments, the bandwidth available between the client and Level-1 nodes is greater than that between the client and the center. Thus, the center always decided to perform decentralized stage-in. In the next experiment, we modified the setup to use a faster node as the end user site, and repeated the experiment for staging a 2.1 GB file. First, we do the transfer without considering direct transfer and always using decentralized stage-in. Second, we repeat the experiment with the ability to choose between direct and decentralized stage-in depending on the ability to meet a transfer deadline. We observed that for the first case, the time to stage and transfer the data to the center was 2867 seconds. In contrast, for the second case the direct transfer completed in 968 seconds, an improvement of 66.2%. This stresses the need for the stage-in mechanisms to dynamically adjust to the variations in the system behavior, and to not be hard-wired to simply always do a staged transfer or a direct transfer.

VI. Conclusion

In this paper, we have presented the design and a proof-of-concept implementation of a just-in-time staging framework to coincide input data delivery at the supercomputing center scratch space with job startup time. Our framework leverages the job wait time estimates from a batch queue prediction service and user-specified intermediate nodes to deliver input data in time. We use our approach in conjunction with the BitTorrent protocol, instrumented to use dynamic network monitoring information, to adapt to transient network conditions and to tap available residual network bandwidth between participants. Moreover, our prototype has been integrated with the PBS scheduler and our own data queue/dependency management tool, `stagesub`, that is being used in the Jaguar supercomputer at ORNL. Our evaluation indicates that our framework can stage the input data only when needed and can thus optimize center scratch usage, improve center serviceability, and protect input data from undesirable failure scenarios.

References

[1] Batch Queue Prediction, September 2008. http://nws.cs.ucsb.edu/ewiki/nws.php?id=Batch+Queue+Prediction.

[2] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. November 1999. http://www-unix.mcs.anl.gov/openpbs/docs/v2_2_ers.pdf.

[3] B. Cohen. BitTorrent Protocol Specification, May 2007. http://www.bittorrent.org/protocol.html.

[4] A. Downey. Using queue time predictions for processor allocation. In *Proc. JSSPP*, 1997.

[5] Druschel et. al. Freepastry, 2004. http://freepastry.rice.edu/.

[6] Grid Infrastructure Group. TeraGrid, May 2007. http://www.teragrid.org/.

[7] C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *SC*, 2005.

[8] P. Maymounkov. Online Codes. Technical Report TR2003-883, New York University, November 2002.

[9] H. M. Monti, A. R. Butt, and S. S. Vazhkudai. Timely offbading of result-data in hpc centers. In *Proc. ICS*, 2008.

[10] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. ACM HotNets*, 2002.

[11] J. S. Plank. Erasure codes for storage applications, 2005. Tutorial Slides, presented at *USENIX FAST*, http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html.

[12] W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Proc. JSSPP*, 1997.

[13] S. Vazhkudai and X. Ma. Recovering transient data: Automated on-demand data reconstruction and offbading on supercomputers. *ACM SIGOPS Operating Systems Review: Special Issue on File and Storage Systems*, 41(1):14–18, 2007.

[14] S. Vazhkudai and J. Schopf. Predicting sporadic grid data transfers. In *Proc. HPDC*, 2002.

[15] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems*, 15(5):757–768, 1999.

[16] Z. Zhang, C. Wang, S. S. Vazhkudai, X. Ma, G. Pike, J. Cobb, and F. Mueller. Optimizing center performance through coordinated data staging, scheduling and recovery. In *Proc. SC*, 2007.